

LA-UR-04-3024

*Approved for public release;  
distribution is unlimited.*

*Title:* Benchmarking Small Matrix Multiplication on  
Various Fortran Compilers and Platforms

*Author(s):* Jobie M. Gerken

*Submitted to:*



Los Alamos National Laboratory, an affirmative action/equal opportunity employer, is operated by the University of California for the U.S. Department of Energy under contract W-7405-ENG-36. By acceptance of this article, the publisher recognizes that the U.S. Government retains a nonexclusive, royalty-free license to publish or reproduce the published form of this contribution, or to allow others to do so, for U.S. Government purposes. Los Alamos National Laboratory requests that the publisher identify this article as work performed under the auspices of the U.S. Department of Energy. Los Alamos National Laboratory strongly supports academic freedom and a researcher's right to publish; as an institution, however, the Laboratory does not endorse the viewpoint of a publication or guarantee its technical correctness.

# Benchmarking Small Matrix Multiplication on Various Fortran Compilers and Platforms

Jobie M. Gerken  
Los Alamos National Laboratory  
`jgerken@lanl.gov`

April, 19 2004

## Abstract

The efficiency of square matrix multiplication for matrices from  $2 \times 2$  to  $18 \times 18$  is reported for five algorithms on several architecture, operating system and compiler combinations. The algorithms include: (a) intrinsic `matmul`; (b) two `DO` loops with the intrinsic `dot_product`; (c) three `DO` loops; (d) BLAS Level 3 `dgemm`; and, (e) `FORALL` loop with the intrinsic `dot_product`. The architecture, operating system and compiler combinations include: (a) HP/Compaq, TRU, HP; (b) SGI, IRIX, MIPSPro; (c) SGI, IRIX, NAGWare; (d) Compaq, Linux, Absoft; (e) Compaq, Linux, Intel; (f) Compaq, Linux, Lahey; and, (g) Compaq, Linux, Portland Group.

## 1 Matrix Multiply Algorithm

To investigate the efficiency of matrix multiplication the following general algorithm was used:

```
allocate(A(dim,dim),B(dim,dim),C(dim,dim))
call cpu_time(time_start)
do n=1,10000
A(:,1) = A(:,1)*random_number
[C] = [A][B]
write(12)C
enddo
call cpu_time(time_finish)
total_time = time_finish - time_start
```

Where  $[C] = [A][B]$  is the matrix multiplication operation and is one of:

1. Intrinsic `matmul`

```
c = matmul(a,b)
```

2. Two `DO` loops with intrinsic `dot_product`

```
do i = 1, ndim
  do j = 1, ndim
    c(i,j) = dot_product(a(i,:),b(:,j))
  enddo
enddo
```

3. Three DO loops

```
do i = 1, ndim
  do j = 1, ndim
    do k = 1, ndim
      c(i,j) = c(i,j) + a(i,k)*b(k,j)
    enddo
  enddo
enddo
```

4. BLAS Level 3

```
call dgemm('N','N',ndim, ndim, ndim, 1.D0,a,ndim,b,ndim,0.D0,c,ndim)
```

5. FORALL loop with intrinsic dot\_product

```
forall(i=1:ndim,j=1:ndim)c(i,j) = dot_product(a(i,:),b(:,j))
```

Each loop method is independent of the order of the indices. For methods 2 and 3 each possible loop variation is reported, however, for method 5, only the variation shown above is reported. The program source code showing all variations that were tested is included at the end of this report.

## 2 Architectures, Operating Systems and Compilers

1. QSC

- (a) Architecture: HP/Compaq ES-45, 1.25 GHz Alpha EV6
- (b) Operating System: TRU64
- (c) Compiler: HP/Compaq Fortran V5.4A-1472

2. Theta

- (a) Architecture: SGI Origin 2000, 250 MHz IP27 R10000 rev. 3.4
- (b) Operating System: IRIX64
- (c) Compiler: SGI MIPSPro V7.3.1.2m
- (d) Compiler: NAGWare Fortran 95 V4.2

3. Columbia

- (a) Architecture: SGI Onyx2, 195 MHz IP27 R10000 rev. 2.6
- (b) Operating System: IRIX64
- (c) Compiler: SGI MIPSPro V7.30

4. Lambda

- (a) Architecture: Compaq DL360, 1 GHz Intel Pentium 3
- (b) Operating System: Red Hat Linux 2.4.18 kernel
- (c) Compiler: Absoft Pro V7.5
- (d) Compiler: Absoft Pro V8.0
- (e) Compiler: Absoft Pro V8.2
- (f) Compiler: Intel V6.0
- (g) Compiler: Intel V7.0
- (h) Compiler: Lahey Fortran 95 V6.0
- (i) Compiler: Lahey Fortran 95 Pro V6.2
- (j) Compiler: Portland Group V5.1

### 3 Results

CPU times are presented in the following figures. Each figure contains two plots, with the first being the results for matrices from  $2 \times 2$  to  $8 \times 8$ , and the second plot being the entire range of results. Also included in each figure are the times for the general algorithm without the matrix multiplication to give a measure of the *algorithmic constant* time (i.e. the time spent doing everything except multiplication). The title of each figure includes the machine name, CPU type, operating system and compiler, as well as the command used to compile the program. The results for the Portland Group compiler contain two figures. The first is a single plot showing all results, the second figure shows the two plot format scaled to exclude the results for the `FORALL` loop with the intrinsic `dot_product`.

All of the algorithms for matrix multiplication are in standard Fortran except the BLAS routine `dgemm` which is called from an external library. A library implementation was linked with the executable for all but the NAGWare and Intel compilers. Where the figures for these two compilers show `dgemm` results, these are in reality the results with the call to `dgemm` commented out and hence the results are similar to the no multiply results.

The naming convention in the legends uses  $i$ ,  $j$ , and  $k$  to indicate DO loop order, where the inner loop instruction is  $c(i,j) = \text{dot\_product}(a(i,:), b(:,j))$  or  $c(i,j) = c(i,j) + a(i,k)*b(k,j)$ ,  $dp$  indicates the intrinsic `dot_product`,  $matmul$  indicates the intrinsic `matmul`,  $dgemm$  indicates the BLAS Level 3 routine `dgemm`, `forall` indicates the `FORALL` loop with the intrinsic `dot_product`, and *no mult* indicates the algorithmic constant time.

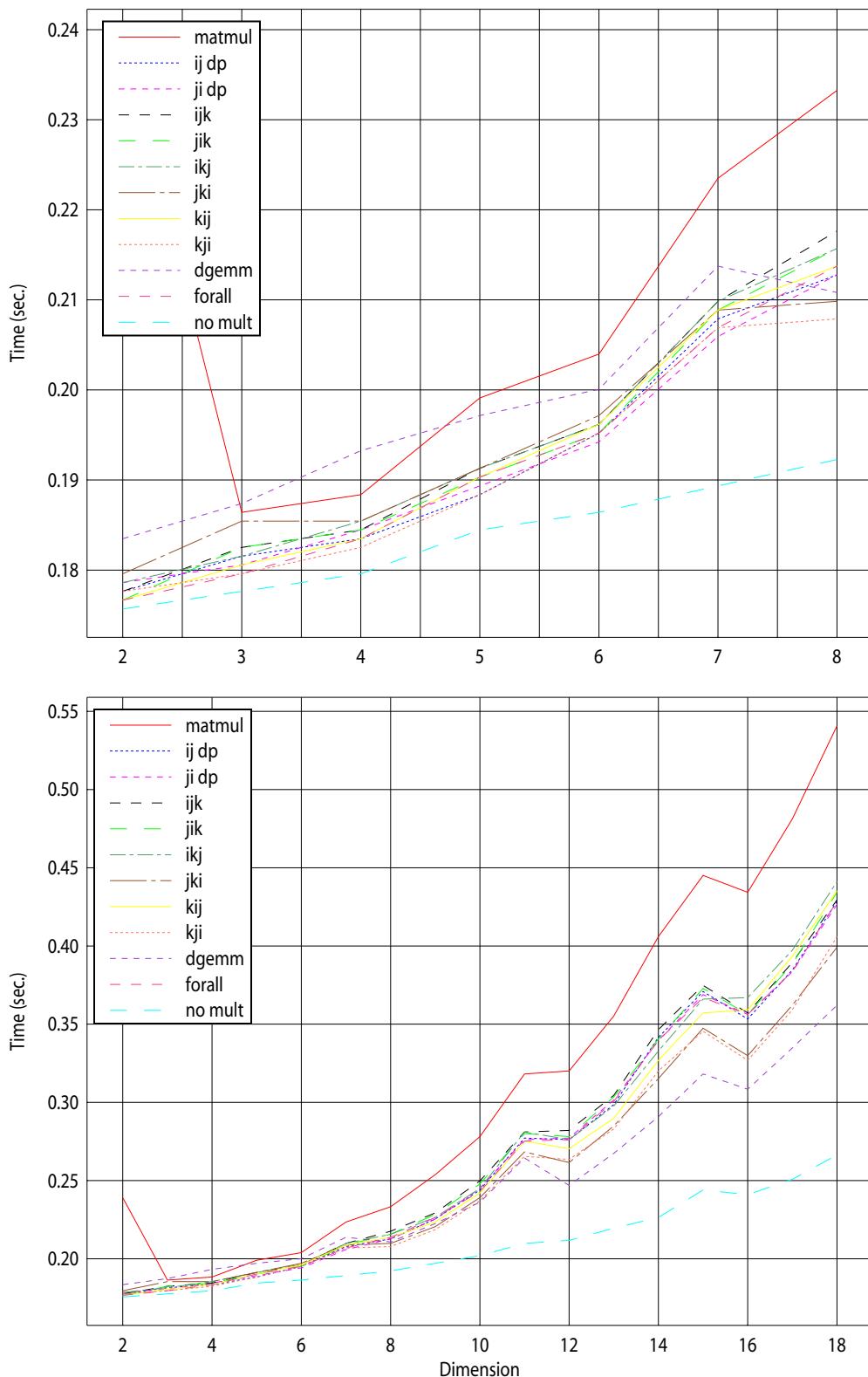


Figure 1: QSC, 1.25 GHz Alpha, TRU64, HP V5.4  
f90 -fast matmul\_alloc\_small.f90 -lcxml -o xmatmul\_alloc\_small

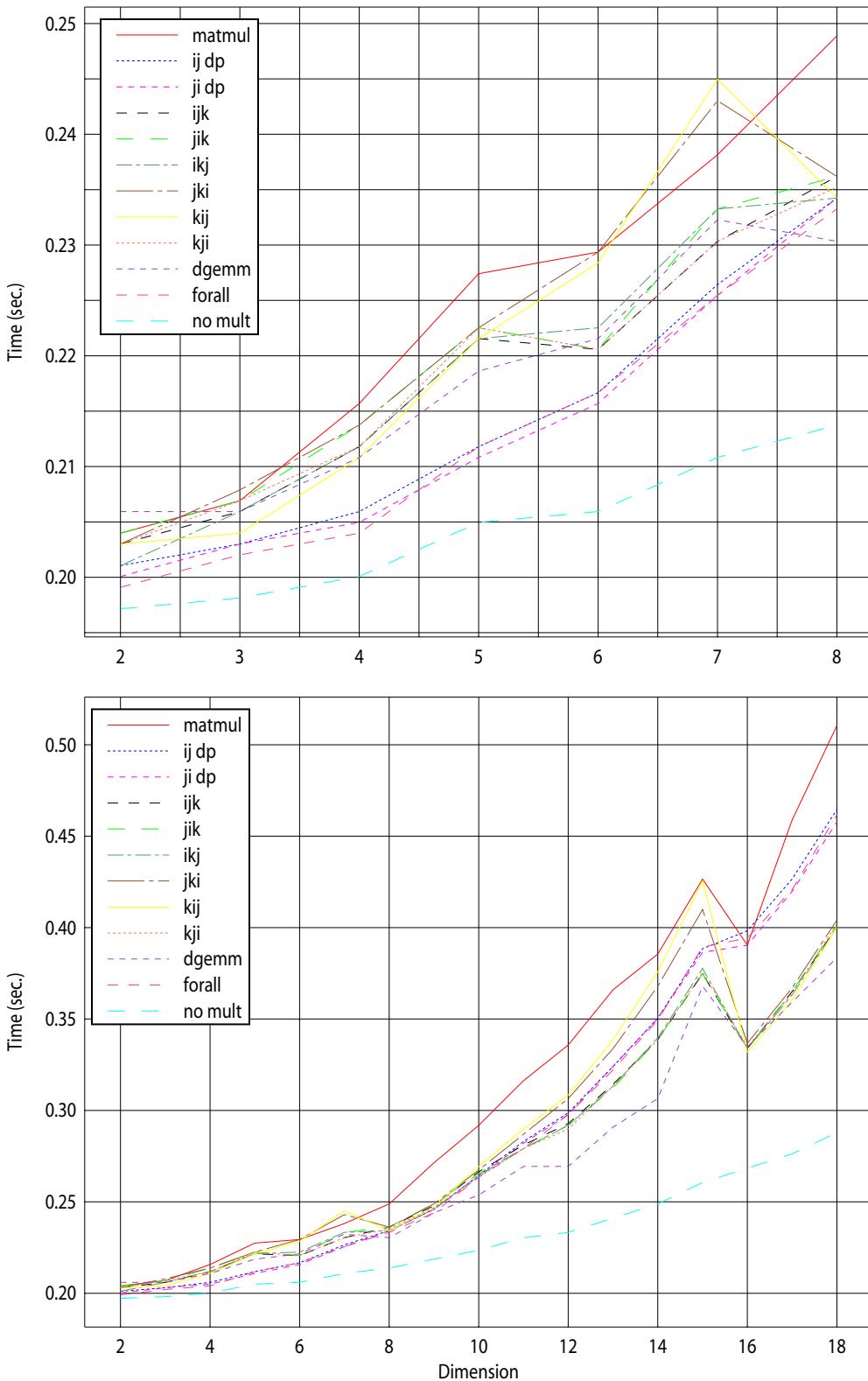


Figure 2: QSC, 1.25 GHz Alpha, TRU64, HP V5.4

f90 -fast -assume nobigarray -O5 -pipeline matmul\_alloc\_small.f90 -lcxml -o xmatmul\_alloc\_small

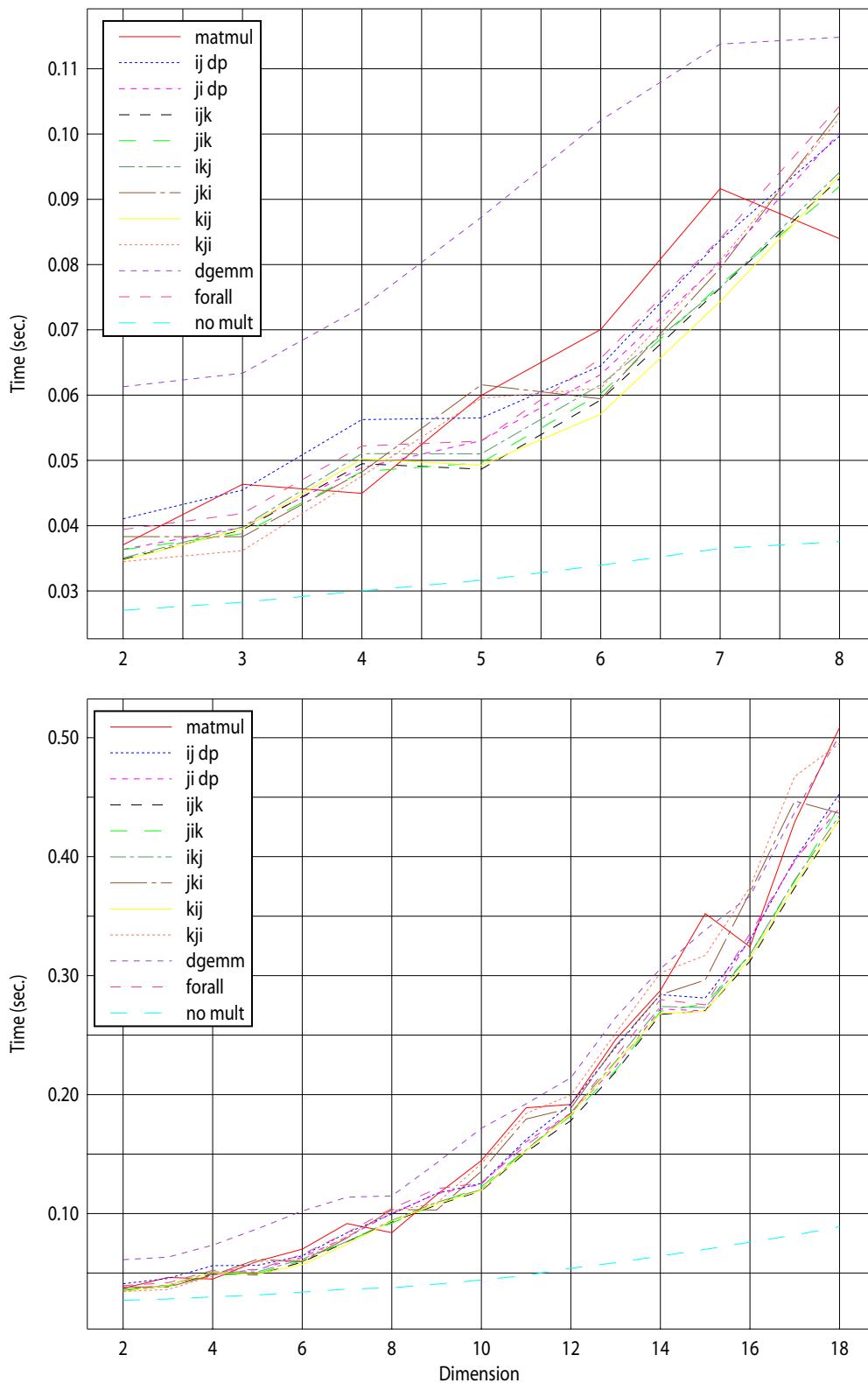


Figure 3: Theta, 250 MHz R10000, IRIX64, MIPSPro V7.3  
f90 -O3 -mips4 -64 -r10000 matmul\_alloc\_small.f90 -lblas -o xmatmul\_alloc\_small

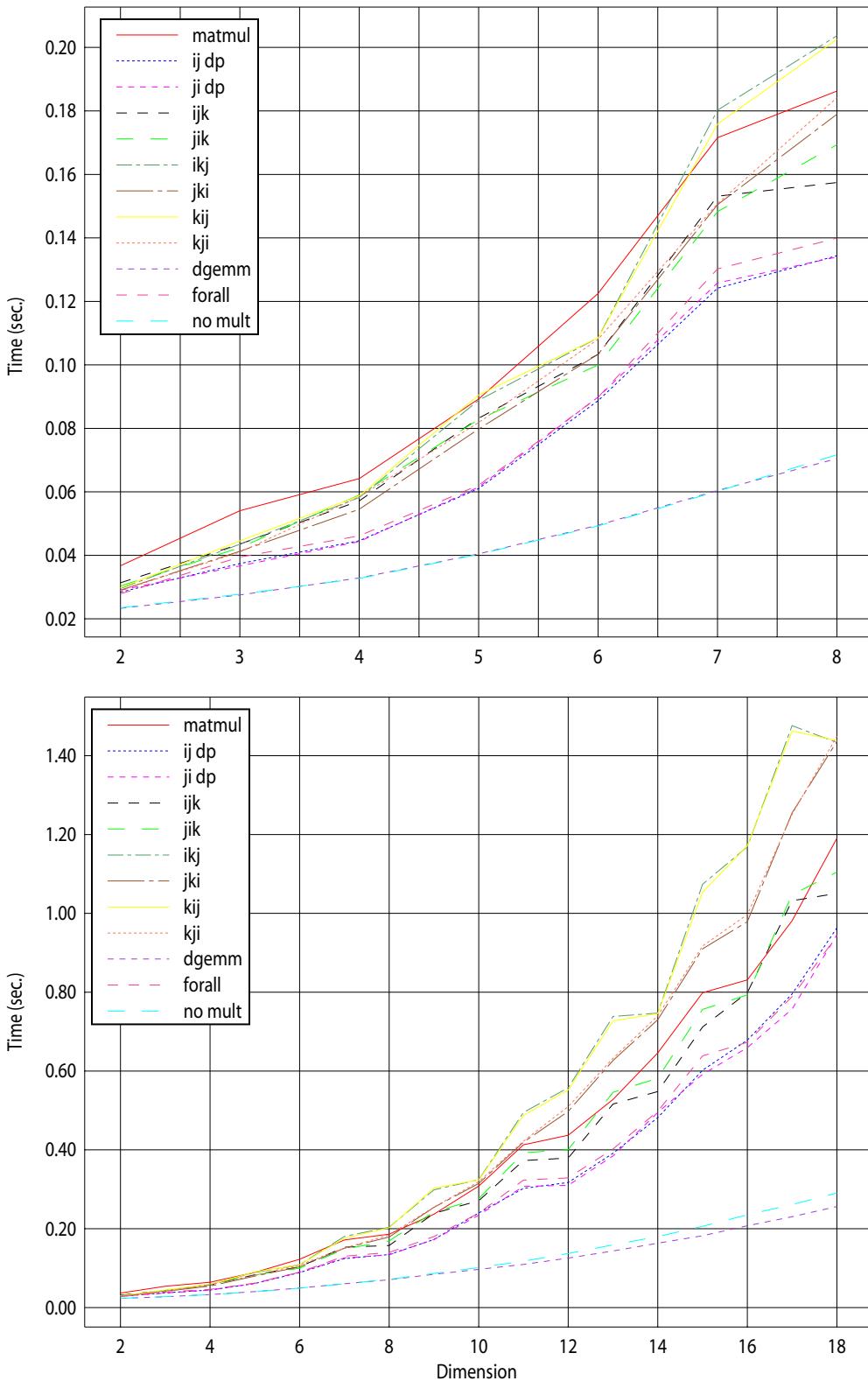


Figure 4: Theta, 250 MHz R10000, IRIX64, NAGWare V4.2  
f95 -abi=64 -O4 -target=mips4 matmul\_alloc\_small.f90 -o xmatmul\_alloc\_small

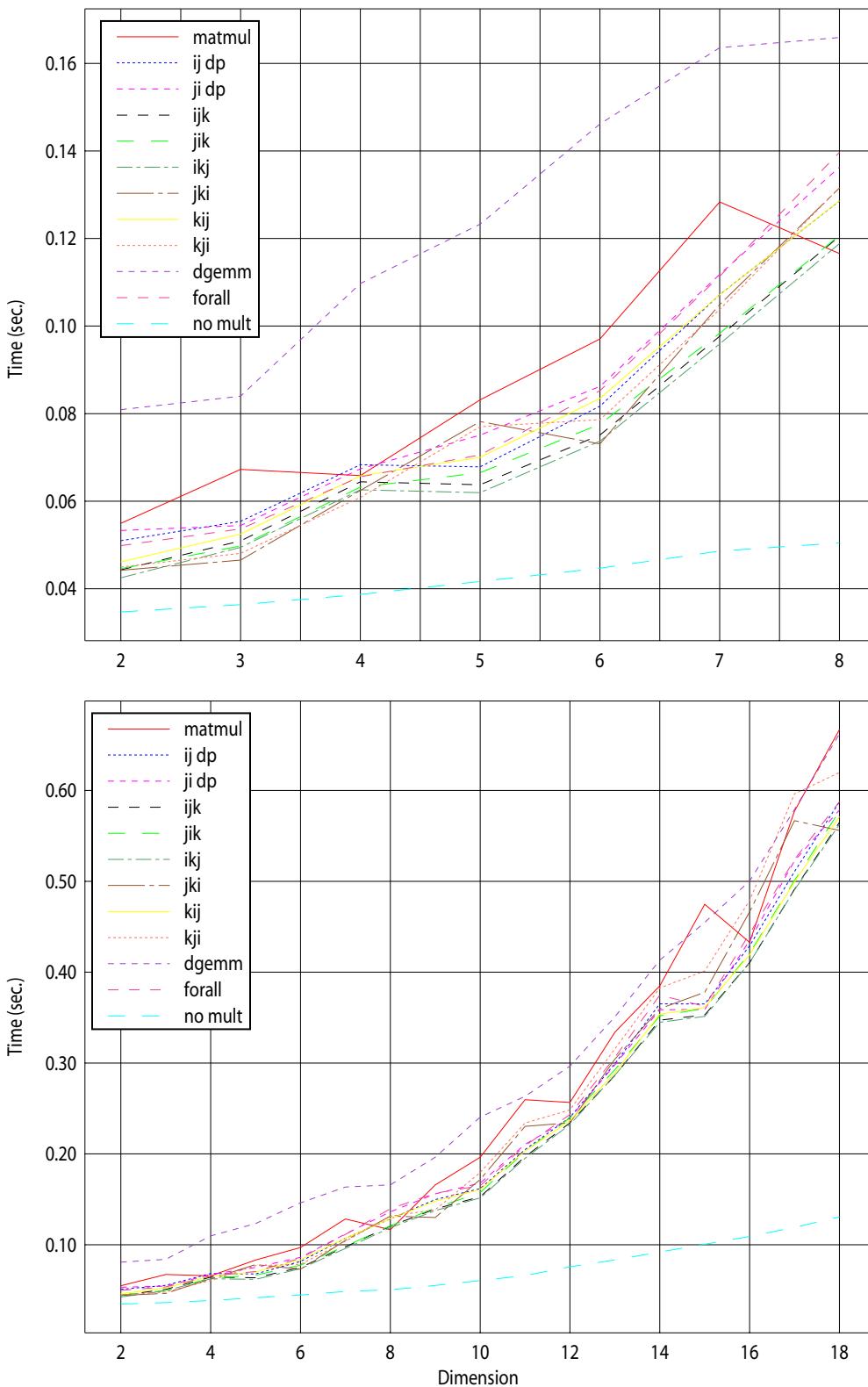


Figure 5: Columbia, 195 MHz R10000, IRIX64, MIPSPro V7.3  
f90 -O3 -mips4 -64 -r10000 matmul\_alloc\_small.f90 -lblas -o xmatmul\_alloc\_small

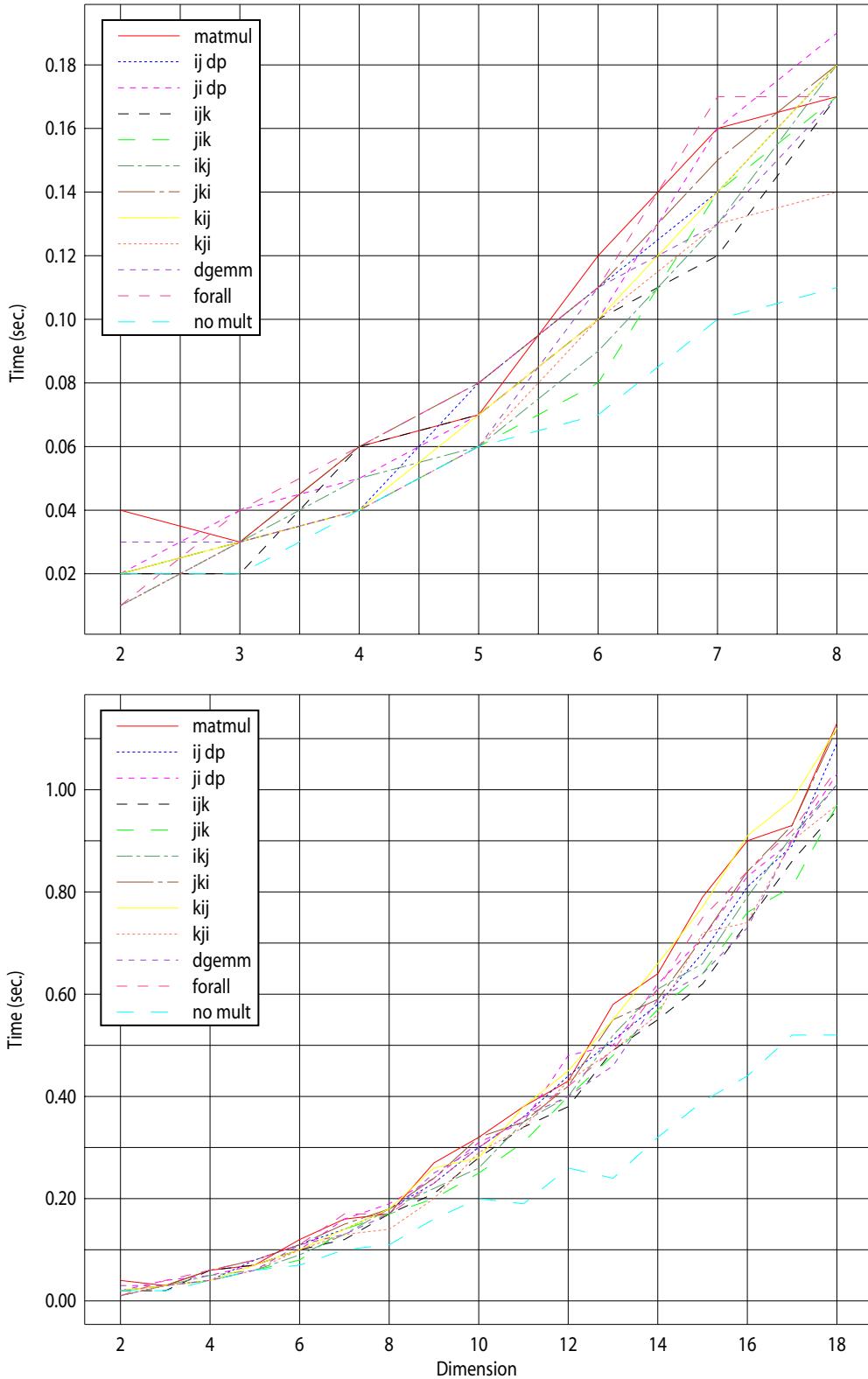


Figure 6: Lambda, 1 GHz Pentium 3, Red Hat Linux, Absoft V7.5  
f90 -limslblas -limsl -YEXT\_NAMES="LCS" -YEXT\_SFX=\_ -O2 -lU77  
-cpu:p6 matmul\_alloc\_small.f90 -o xmatmul\_alloc\_small

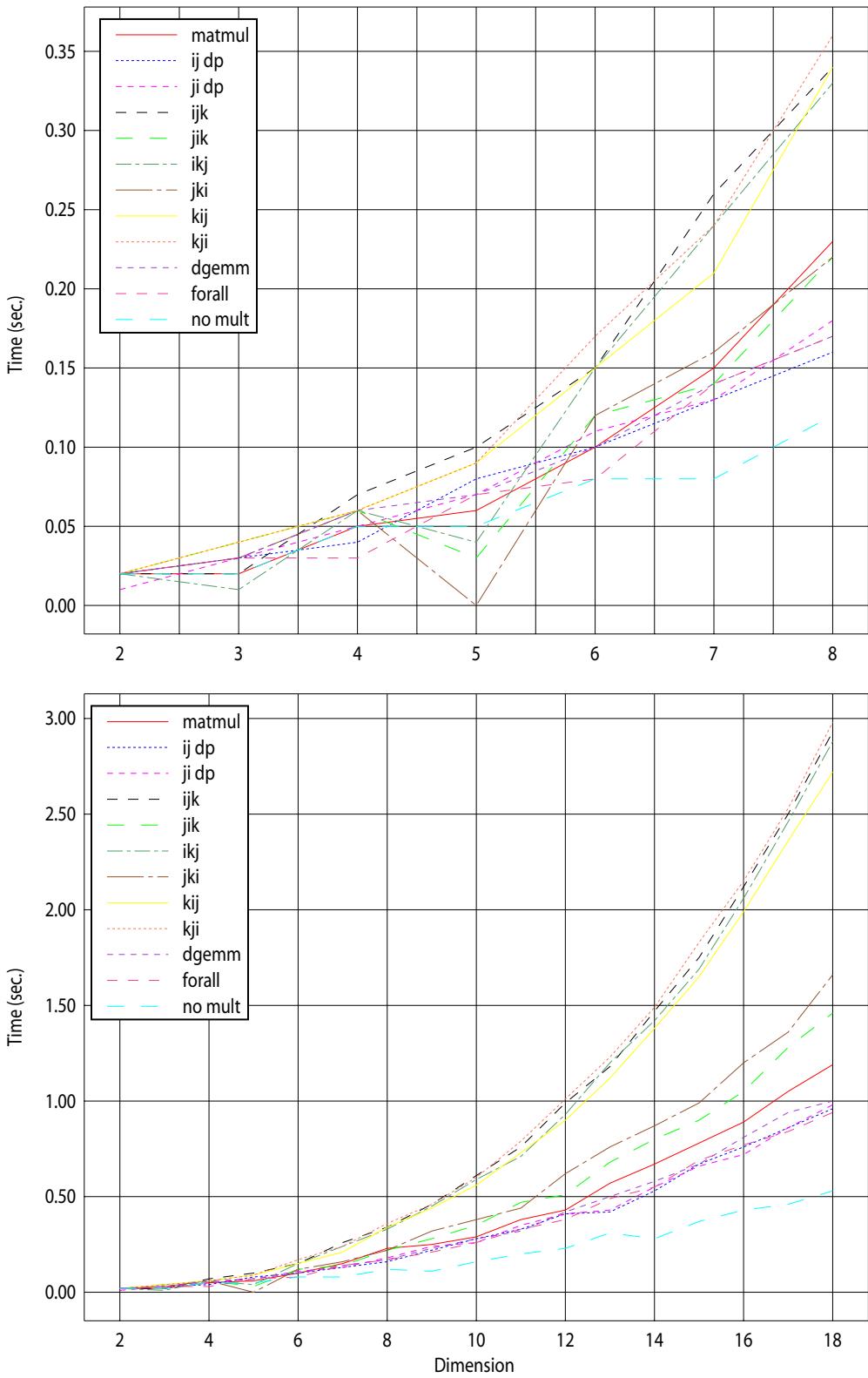


Figure 7: Lambda, 1 GHz Pentium 3, Red Hat Linux, Abssoft V8.0  
f90 -lblas -YEXT\_NAMES="LCS" -YEXT\_SFX="\_" -O2 -lU77 -cpu:p6 matmul\_alloc\_small.f90 -o xmatmul\_alloc\_small

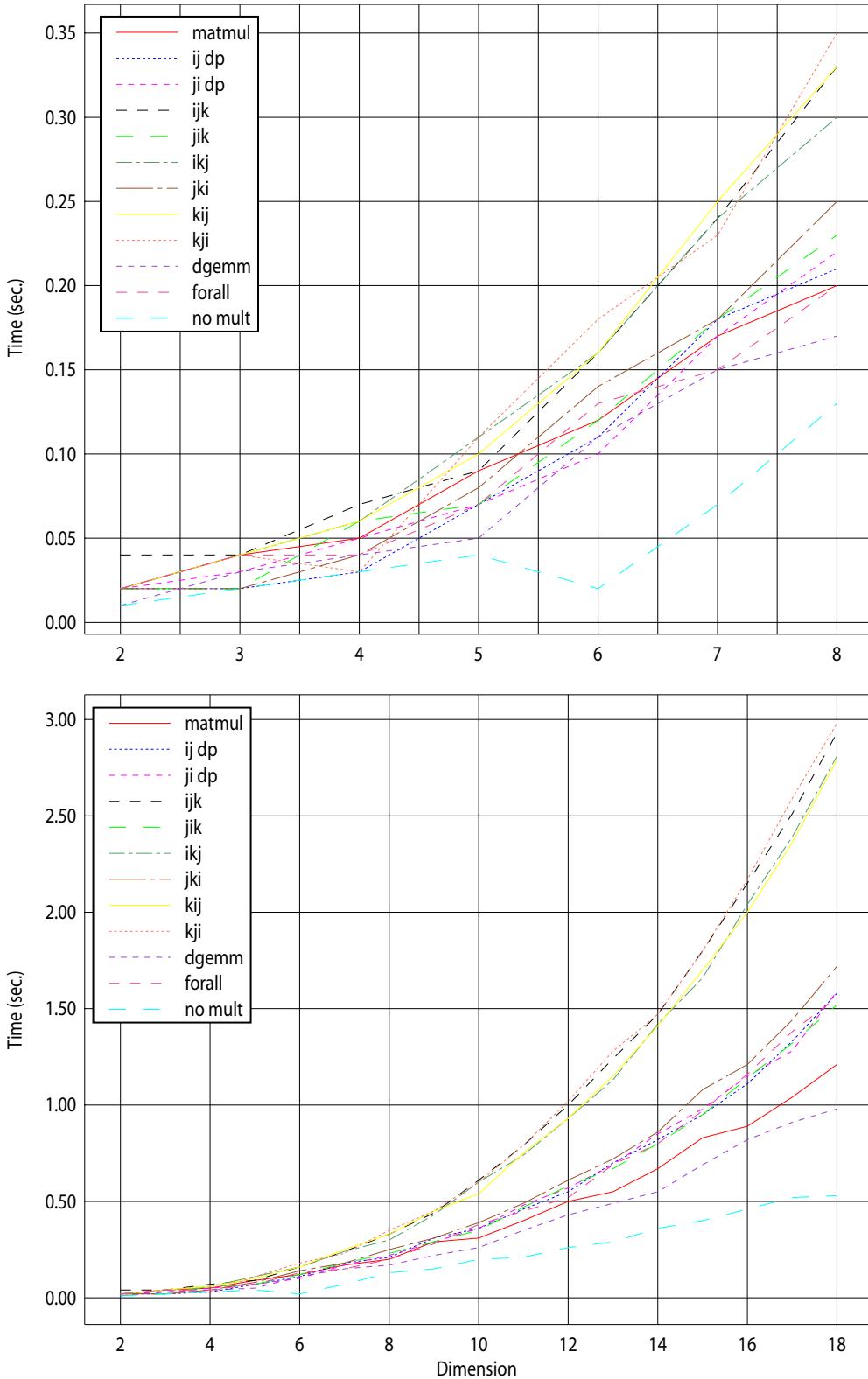


Figure 8: Lambda, 1 GHz Pentium 3, Red Hat Linux, Absoft V8.2  
`f90 -L/usr/local/packages/absoft-pro-8.0/opt/absoft/lib -lblas -YEXT_NAMES="LCS"  
-YEXT_SFX=_ -O3 -lU77 -cpu:p6 matmul_alloc_small.f90 -o xmatmul_alloc_small`

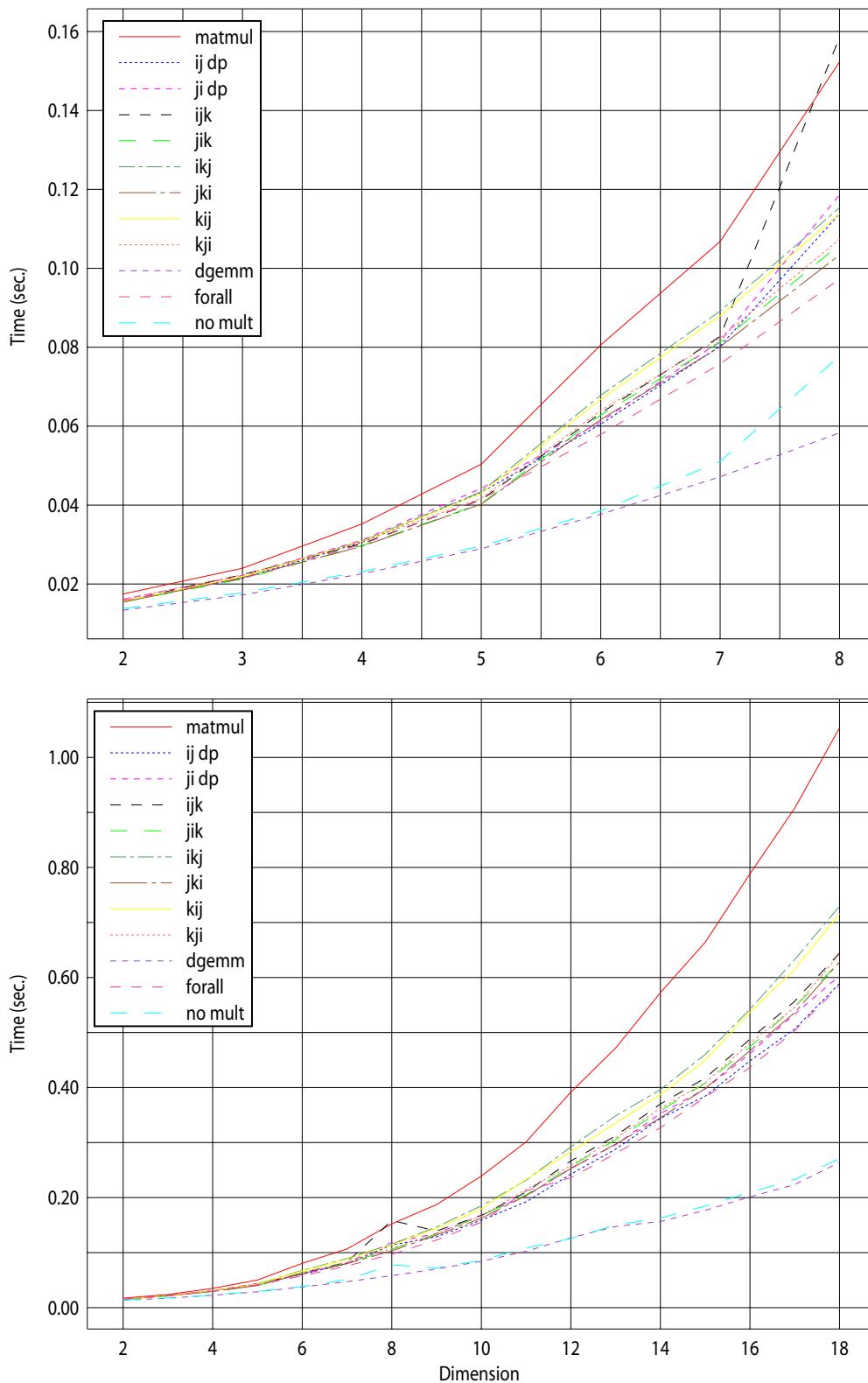


Figure 9: Lambda, 1 GHz Pentium 3, Red Hat Linux, Intel V6.0  
ifc -O3 -tpp6 -align matmul\_alloc\_small.f90 -o xmatmul\_alloc\_small

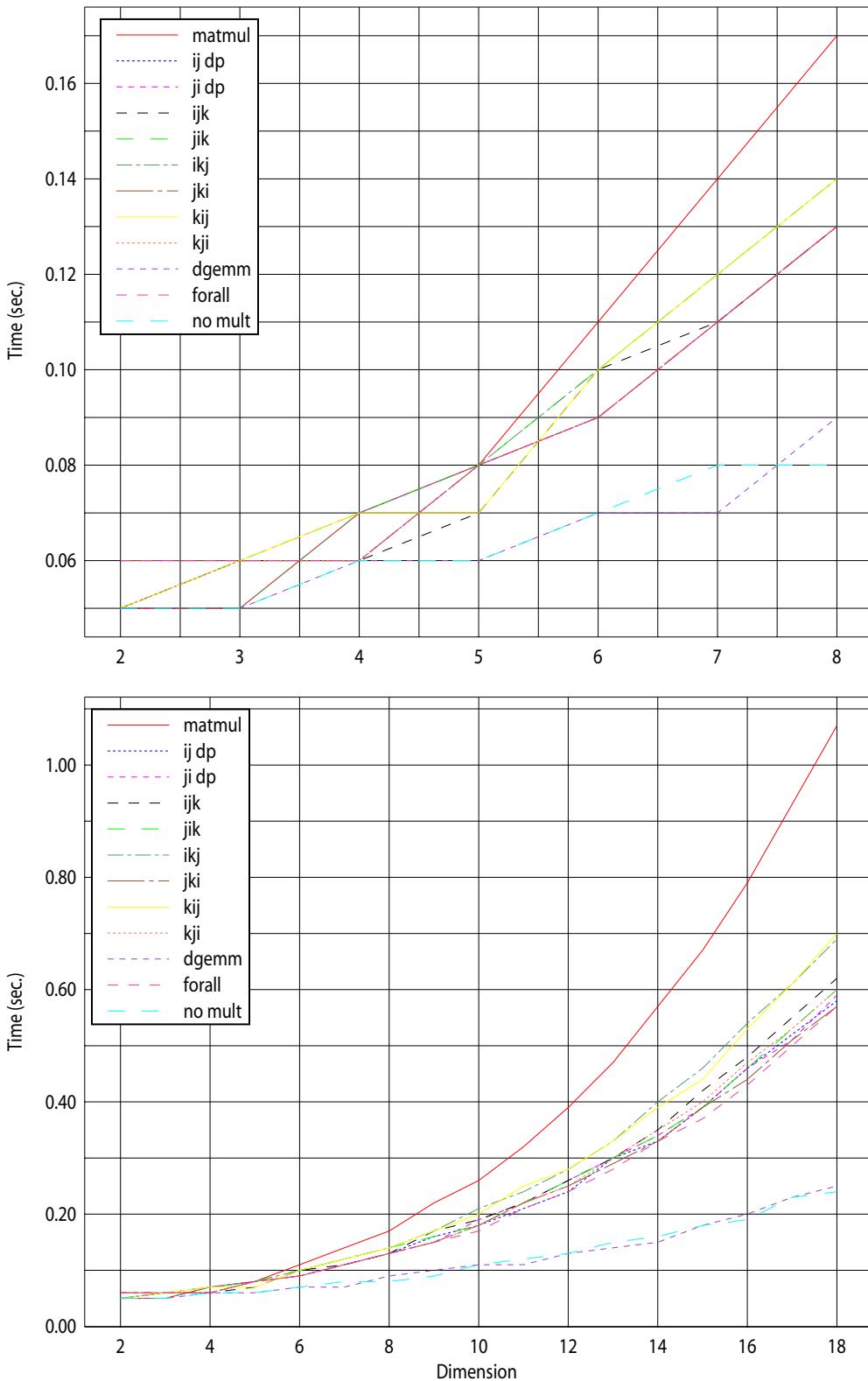


Figure 10: Lambda, 1 GHz Pentium 3, Red Hat Linux, Intel V7.0  
ifc -O3 -tpp6 -align matmul\_alloc\_small.f90 -o xmatmul\_alloc\_small

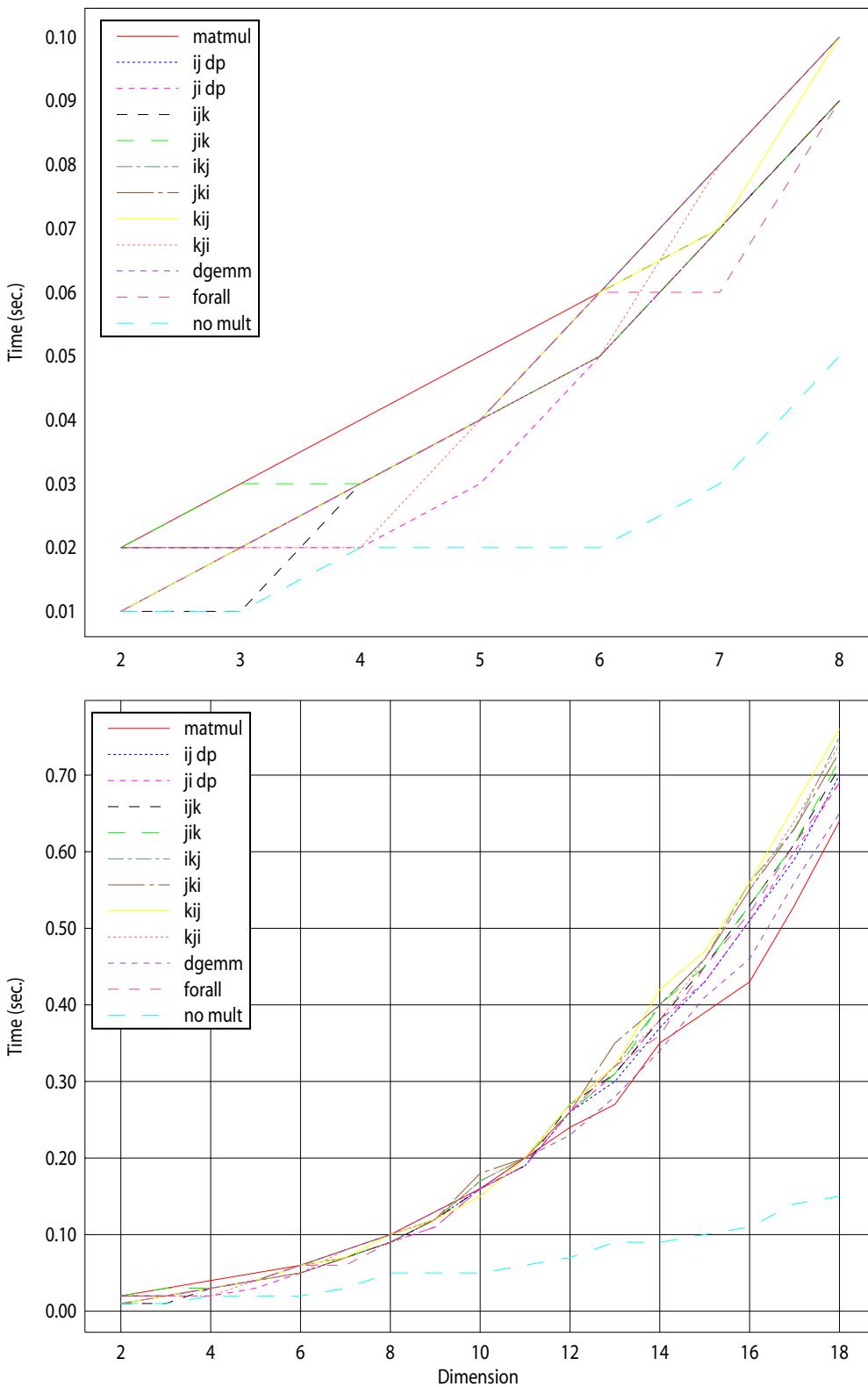


Figure 11: Lambda, 1 GHz Pentium 3, Red Hat Linux, Lahey V6.0  
`lf95 -O -tpp matmul_alloc_small.f90 -L/usr/lib/gcc-lib/i386-redhat-linux/egcs-2.91.66/ -lblas -lg2c -o xmatmul_alloc_small`

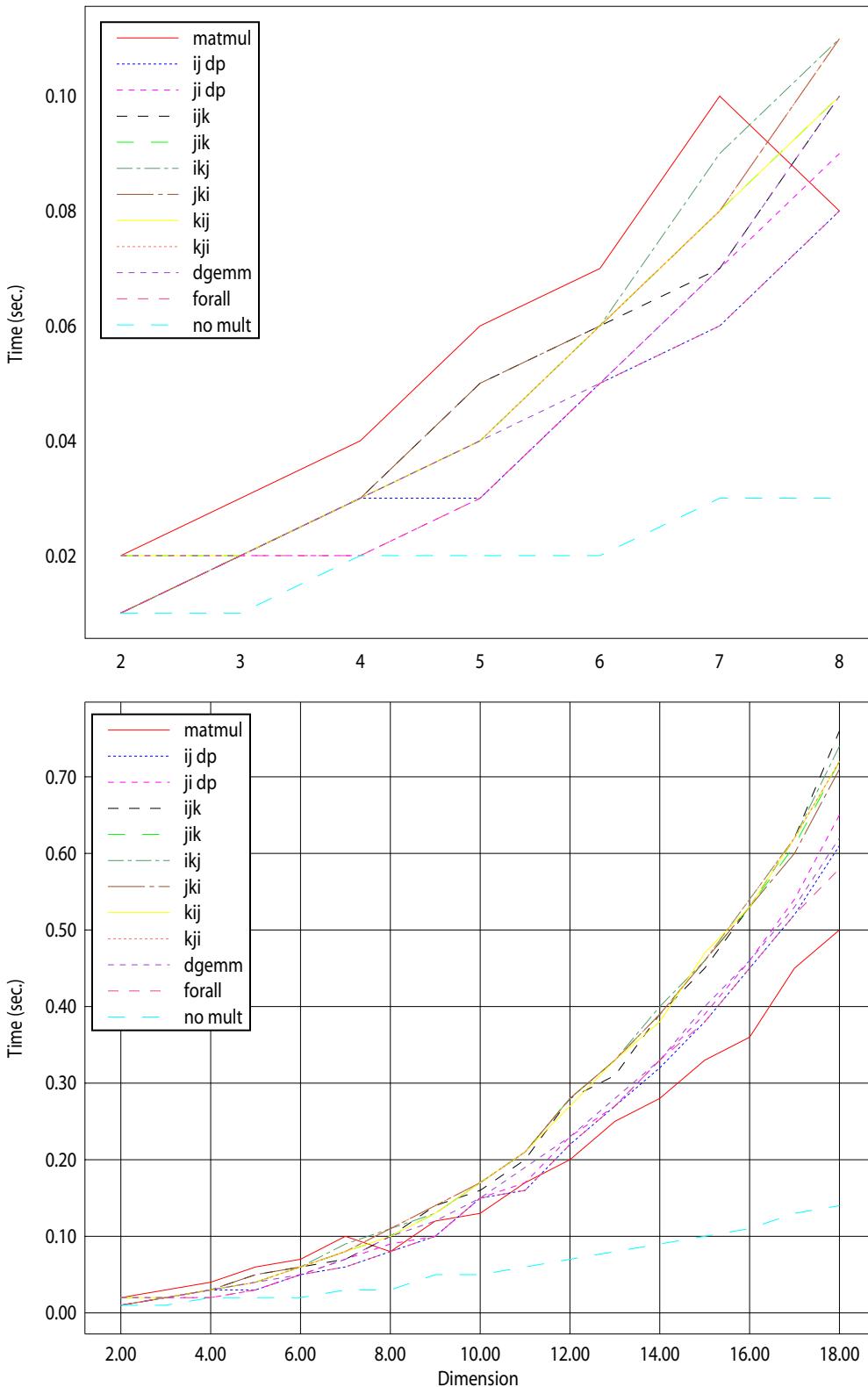


Figure 12: Lambda, 1 GHz Pentium 3, Red Hat Linux, Lahey Pro V6.2  
`if95 -O -tpp matmul_alloc_small.f90 -L/usr/lib/gcc-lib/i386-redhat-linux/egcs-2.91.66/ -lblas -lg2c -o xmatmul_alloc_small`

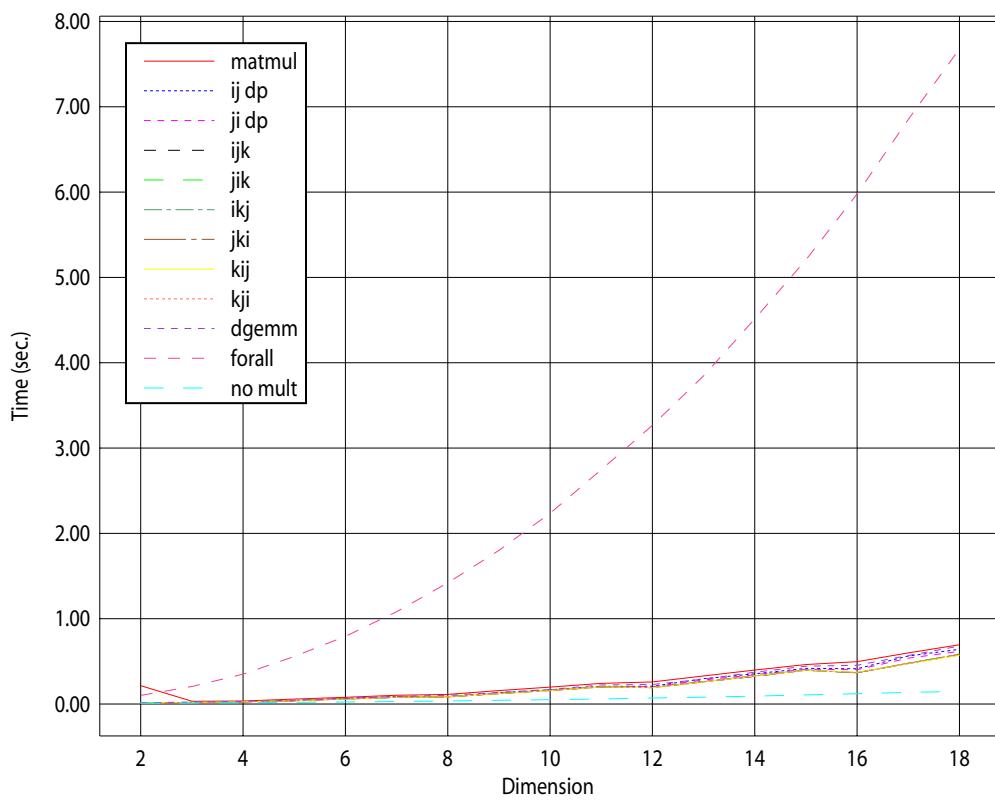


Figure 13: Lambda, 1 GHz Pentium 3, Red Hat Linux, Portland Group V5.1  
`pgf90 -fastsse matmul_alloc_small.f90 -lblas -o xmatmul_alloc_small`

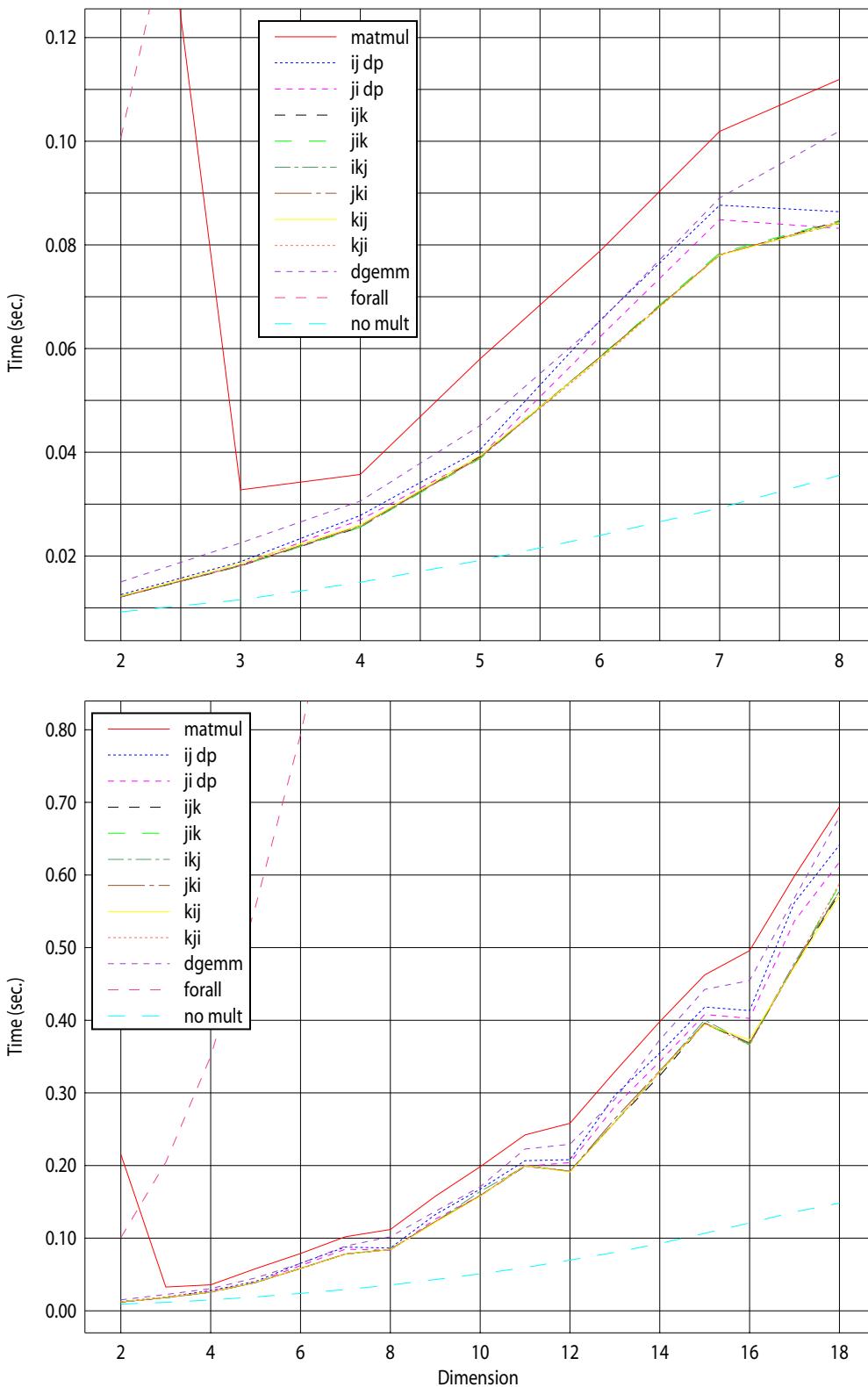


Figure 14: Lambda, 1 GHz Pentium 3, Red Hat Linux, Portland Group V5.1  
`pgf90 -fastsse matmul_alloc_small.f90 -lblas -o xmatmul_alloc_small`

## 4 Fortran Program

```
program matmul_small
implicit none
INTEGER,PARAMETER :: IKIND= SELECTED_INT_KIND(8)
INTEGER,PARAMETER :: DPKIND= SELECTED_REAL_KIND(15,200)
integer(IKIND),parameter :: navg=1,nmult=10000
integer(IKIND),parameter :: dim0=2,dimf=18,strid=1
integer(IKIND) :: i, j, k,m,n, ierr, ndim
real(DPKIND), allocatable :: a(:, :, :), b(:, :, :), c(:, :, :)
real(DPKIND),dimension(2) :: tarray1
real(SELECTED_REAL_KIND(5,20)) :: etime
real(DPKIND) :: tt(2), all_tt(12)
real :: rand(nmult)
EXTERNAL dgemm

call random_seed()
call random_number(rand)
open(unit=12,file="/scratch/gerken/junk_small",form="unformatted")
open(unit=11,file="times_alloc_small")
write(11,'(13A14)') 'ndim','matmul','ij_dotprod','ji_dotprod',&
'ijk','jik','ikj','jki','kij','kji','dgemm','forall', 'no_mult'
!-----
do ndim=dim0,dimf,strid
all_tt(:) = 0._DPKIND
allocate(a(ndim,ndim),b(ndim,ndim),c(ndim,ndim),stat=ierr)
if(ierr/=0)then
  write(*,*)"Allocation error, ndim=",ndim
  stop
endif
do j = 1, ndim
  do i = 1, ndim
    a(i,j) = (i+j)*1._DPKIND
  enddo
enddo
b = a
do m=1,navg
  a(:,ndim) = a(:,ndim)*rand(min(m,nmult))
!-----
call cpu_time(tt(1))
do n=1,nmult
  a(:,1) = a(:,1)*rand(n)
  c = matmul(a,b)
  write(12)c
enddo
call cpu_time(tt(2))
all_tt(1) = all_tt(1) + (tt(2) - tt(1))/(1.*navg)
!-----
call cpu_time(tt(1))
do n=1,nmult
  a(:,1) = a(:,1)*rand(n)
  do i = 1, ndim
    do j = 1, ndim
      c(i,j) = dot_product(a(i,:),b(:,j))
    enddo
  enddo
enddo
```

```

        enddo
    enddo
    write(12)c
    enddo
    call cpu_time(tt(2))
    all_tt(2) = all_tt(2) + (tt(2) - tt(1))/(1.*navg)
!
-----call cpu_time(tt(1))
do n=1,nmult
a(:,1) = a(:,1)*rand(n)
do j = 1, ndim
    do i = 1, ndim
        c(i,j) = dot_product(a(i,:),b(:,j))
    enddo
enddo
write(12)c
enddo
call cpu_time(tt(2))
all_tt(3) = all_tt(3) + (tt(2) - tt(1))/(1.*navg)
!
-----call cpu_time(tt(1))
do n=1,nmult
c(:,:)=0.0_DPKIND
a(:,1) = a(:,1)*rand(n)
do i = 1, ndim
    do j = 1, ndim
        do k = 1, ndim
            c(i,j) = c(i,j) + a(i,k)*b(k,j)
        enddo
    enddo
enddo
write(12)c
enddo
call cpu_time(tt(2))
all_tt(4) = all_tt(4) + (tt(2) - tt(1))/(1.*navg)
!
-----call cpu_time(tt(1))
do n=1,nmult
c(:,:)=0.0_DPKIND
a(:,1) = a(:,1)*rand(n)
do j = 1, ndim
    do i = 1, ndim
        do k = 1, ndim
            c(i,j) = c(i,j) + a(i,k)*b(k,j)
        enddo
    enddo
enddo
write(12)c
enddo
call cpu_time(tt(2))
all_tt(5) = all_tt(5) + (tt(2) - tt(1))/(1.*navg)
!
-----call cpu_time(tt(1))
do n=1,nmult

```

```

c(:, :)=0.0_DPKIND
a(:, 1) = a(:, 1)*rand(n)
do i = 1, ndim
  do k = 1, ndim
    do j = 1, ndim
      c(i,j) = c(i,j) + a(i,k)*b(k,j)
    enddo
  enddo
enddo
write(12)c
enddo
call cpu_time(tt(2))
all_tt(6) = all_tt(6) + (tt(2) - tt(1))/(1.*navg)
!-----
call cpu_time(tt(1))
do n=1,nmult
c(:, :)=0.0_DPKIND
a(:, 1) = a(:, 1)*rand(n)
do j = 1, ndim
  do k = 1, ndim
    do i = 1, ndim
      c(i,j) = c(i,j) + a(i,k)*b(k,j)
    enddo
  enddo
enddo
write(12)c
enddo
call cpu_time(tt(2))
all_tt(7) = all_tt(7) + (tt(2) - tt(1))/(1.*navg)
!-----
call cpu_time(tt(1))
do n=1,nmult
c(:, :)=0.0_DPKIND
a(:, 1) = a(:, 1)*rand(n)
do k = 1, ndim
  do i = 1, ndim
    do j = 1, ndim
      c(i,j) = c(i,j) + a(i,k)*b(k,j)
    enddo
  enddo
enddo
write(12)c
enddo
call cpu_time(tt(2))
all_tt(8) = all_tt(8) + (tt(2) - tt(1))/(1.*navg)
!-----
call cpu_time(tt(1))
do n=1,nmult
c(:, :)=0.0_DPKIND
a(:, 1) = a(:, 1)*rand(n)
do k = 1, ndim
  do j = 1, ndim
    do i = 1, ndim
      c(i,j) = c(i,j) + a(i,k)*b(k,j)
    enddo
  enddo
enddo

```

```

    enddo
  enddo
enddo
write(12)c
enddo
call cpu_time(tt(2))
all_tt(9) = all_tt(9) + (tt(2) - tt(1))/(1.*navg)
!-----
call cpu_time(tt(1))
do n=1,nmult
a(:,1) = a(:,1)*rand(n)
call dgemm('N','N',ndim, ndim, ndim,1.D0,a,ndim,b,ndim,0.D0,c,ndim)
write(12)c
enddo
call cpu_time(tt(2))
all_tt(10) = all_tt(10) + (tt(2) - tt(1))/(1.*navg)
!-----
call cpu_time(tt(1))
do n=1,nmult
a(:,1) = a(:,1)*rand(n)
forall(i=1:ndim,j=1:ndim)c(i,j) = dot_product(a(i,:),b(:,j))
write(12)c
enddo
call cpu_time(tt(2))
all_tt(11) = all_tt(11) + (tt(2) - tt(1))/(1.*navg)
!-----
call cpu_time(tt(1))
do n=1,nmult
a(:,1) = a(:,1)*rand(n)
write(12)c
enddo
call cpu_time(tt(2))
all_tt(12) = all_tt(12) + (tt(2) - tt(1))/(1.*navg)
!-----
rewind(12)
enddo
deallocate(a,b,c)
write(11,'(i14,12E14.6)') ndim,all_tt(:)
enddo
close(11)
close(12)
end program matmul_small

```